Administración de Base de Datos

TESH
HUIXQUILUCAN

2. Arquitectura e instalación de SGBD

2.1 Estructura de memoria y procesos de la instancia

Para que un DBMS pueda funcionar, primero se debe hacer una instancia de este. Esta instancia está compuesta principalmente de tres componentes:

- Archivos
- Estructuras de memoria.
- Estructuras de procesos.

Cada una de estas estructuras se puede dividir a su vez en diferentes puntos.

Archivos

1. **Control (ctl)**:
   - **Propósito**: Estos archivos son fundamentales para el sistema de gestión de bases de datos (DBMS). Contienen metadatos que describen la estructura de la base de datos, como la localización de los ficheros de datos, la configuración del DBMS, y el estado actual de la base de datos.
   - **Importancia**: Son esenciales para las operaciones de recuperación y reinicio del sistema, ya que permiten que el DBMS entienda cómo están organizados los datos y cómo deben ser accedidos.
2. **Rollback Segments (rbs)**:
   - **Propósito**: Facilitan la capacidad de deshacer cambios en la base de datos. Guardan copias de la información antes de que sea modificada por las transacciones. En caso de que una transacción falle o sea necesario deshacerla, estos segmentos permiten restaurar los datos a su estado anterior.
   - **Importancia**: Son clave para mantener la integridad de los datos y para permitir el aislamiento entre transacciones, lo que es fundamental en entornos de bases de datos multiusuario.

3. **Redo Log Files (rdo)**:

   - **Propósito**: Estos archivos registran todas las modificaciones realizadas en la base de datos. A diferencia de los archivos de rollback que guardan el estado anterior, los redo log files registran todas las acciones que modifican la base de datos.
   - **Importancia**: Son esenciales para la recuperación de datos después de fallos del sistema. Permiten "repetir" las transacciones y así restaurar la base de datos al estado en que se encontraba antes del fallo.

4. **Data Files (dbf)**:

   - **Propósito**: Son los archivos donde se almacenan efectivamente los datos de la base de datos. Contienen registros de datos estructurados según los esquemas definidos en el sistema.
   - **Importancia**: Constituyen el núcleo de una base de datos, siendo el lugar donde se guardan tanto los datos de usuario como los datos del sistema.

5. **Index Files (dbi)**:

   - **Propósito**: Estos archivos son estructuras auxiliares que permiten un acceso más rápido a los datos almacenados en los archivos de datos. Un índice puede ser visto como un "directorio" que el DBMS puede usar para localizar rápidamente datos específicos sin tener que leer secuencialmente todo el archivo de datos.
   - **Importancia**: Mejoran significativamente el rendimiento de las consultas, especialmente en bases de datos grandes.

6. **Temporary Files (tmp)**:

   - **Propósito**: Se utilizan para almacenar datos temporales generados durante consultas complejas, operaciones de ordenamiento, y otras tareas que requieren almacenamiento adicional más allá del que está disponible en memoria principal.
   - **Importancia**: Son esenciales para el manejo eficiente de consultas y operaciones que no cabrían en la memoria principal del servidor de base de datos.

Estructura de Memoria

Área Global del sistema (SGA): Es un grupo de estructuras de la memoria compartida que contiene datos e información de control de una instancia de una BD. Si varios usuarios se conectan de forma concurrente a la misma instancia, entonces los datos se comparten en el SGA, por lo que también se llama **shared global area**.

Estructura de Datos del SGA

1. **Caché de los Buffers (Buffer Cache)**:

· **Propósito**: Este componente es parte del System Global Area (SGA) y es responsable de almacenar copias de los bloques de datos leídos desde los archivos de datos en disco. El objetivo es reducir el acceso a disco, que es más lento, manteniendo los datos frecuentemente accesados en una memoria de rápido acceso.

· **Importancia**: Mejora el rendimiento de lectura de la base de datos al reducir el número de lecturas de disco necesarias.

2. **Buffer del registro de Redo (Redo Log Buffer)**:

· **Propósito**: También ubicado en el SGA, este buffer almacena todas las modificaciones realizadas en la base de datos en forma de "registros redo". Estos registros se utilizan para reconstruir cambios en los datos en caso de recuperación tras un fallo.

· **Importancia**: Es esencial para la recuperación de la base de datos, permitiendo que cualquier cambio pueda ser "rehecho" para restaurar el estado de la base de datos después de un fallo.

3. **El Pool compartido (Shared Pool)**:

· **Propósito**: Este es otro componente clave del SGA que almacena datos compartidos como el caché de diccionario de datos, el área de biblioteca, y otros elementos estructurales SQL necesarios para las operaciones de base de datos.

· **Importancia**: Facilita la eficiencia y la velocidad de acceso a la información crucial para el procesamiento de consultas y la gestión de transacciones.

4. **Large Pool**:

   - **Propósito**: Opcional en el SGA, diseñado para aliviar la carga del pool compartido almacenando sesiones de backup de Oracle RMAN, información de multiproceso de servidor, y buffers para ciertas operaciones de gran tamaño.
   - **Importancia**: Optimiza el manejo de la memoria en operaciones grandes y reduce la competencia por recursos en el shared pool.

5. **Java Pool**:

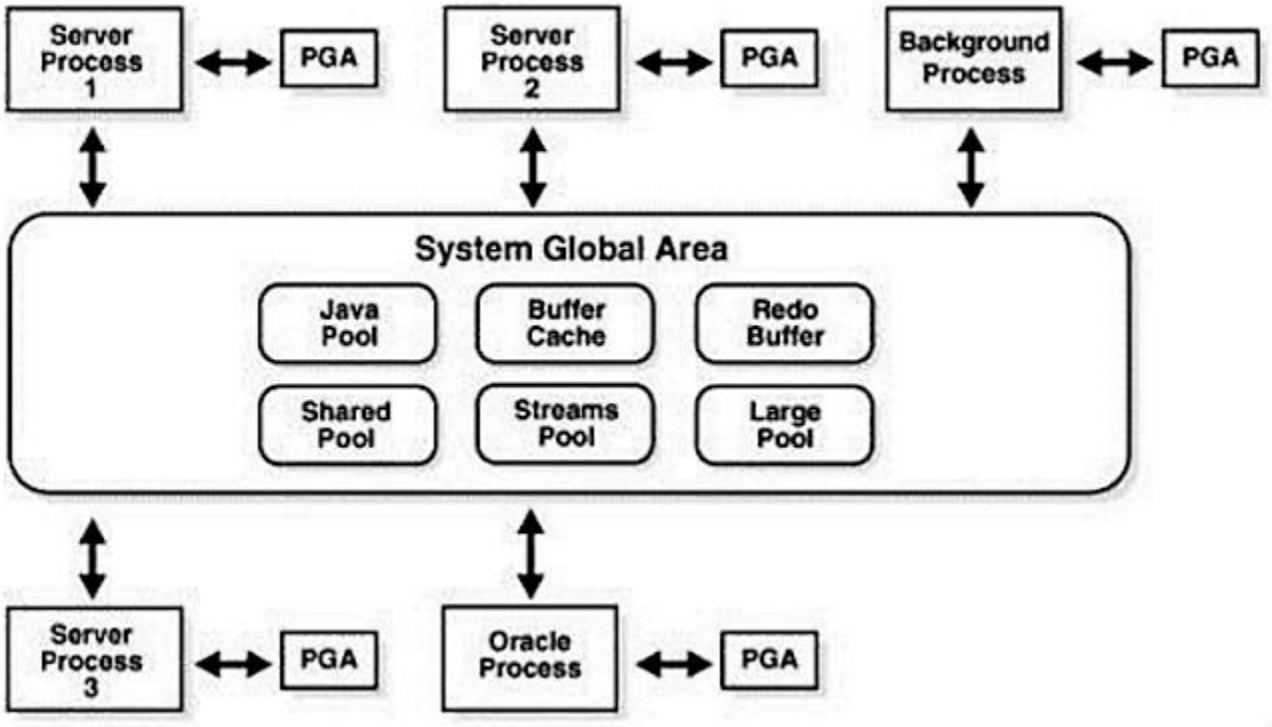
   - **Propósito**: Este pool se utiliza para almacenar datos relacionados con la ejecución de código Java en la base de datos.
   - **Importancia**: Es vital para entornos que ejecutan procedimientos almacenados o desencadenadores escritos en Java, asegurando que haya recursos de memoria dedicados para estas operaciones.

6. **Streams Pool**:

   - **Propósito**: Utilizado en la implementación de Oracle Streams, que facilita la captura de eventos de datos y la propagación de esos eventos a otros sistemas o componentes dentro de una base de datos.
   - **Importancia**: Esencial para el manejo eficiente de eventos de datos y la sincronización de bases de datos en entornos distribuidos.

7. **Caché de diccionario (Dictionary Cache)**:

   - **Propósito**: Parte del shared pool, almacena metadatos sobre la estructura de la base de datos, como información sobre tablas, índices, columnas, usuarios, privilegios y otras construcciones de alto nivel.
   - **Importancia**: Proporciona acceso rápido a esta información crítica, lo cual es esencial para el rendimiento de la base de datos, especialmente durante la compilación y ejecución de consultas.



Estructuras de Proceso

Procesos de usuario: Cada proceso de usuario representa la conexión de un usuario al servidor

Procesos de segundo plano: El servidor se vale de una serie de procesos que son el enlace entre las estructuras físicas y de memoria.

1. **SMON (System Monitor)**:

- **Propósito**: El SMON se encarga de realizar tareas de mantenimiento y recuperación del sistema. Entre sus funciones está la recuperación de instancias tras fallos, limpieza de segmentos temporales no utilizados y la fusión de espacios libres en la base de datos.

- **Importancia**: Esencial para mantener la salud y la estabilidad de la base de datos, asegurando que el sistema se recupere adecuadamente de los fallos y que los recursos sean gestionados eficientemente.

2. **PMON (Process Monitor)**:

- **Propósito**: Supervisa los procesos de la base de datos y se encarga de recuperar los recursos ocupados por procesos fallidos o terminados inesperadamente. También ayuda a limpiar después de sesiones desconectadas y puede resolver problemas de bloqueo de deadlocks.
- **Importancia**: Fundamental para la estabilidad operativa, ya que ayuda a mantener limpio y ordenado el entorno de ejecución.

3. **DBWR (Database Writer)**:

- **Propósito**: Este proceso es responsable de escribir los bloques modificados desde el buffer cache al disco. DBWR escribe los bloques a los archivos de datos y ayuda a gestionar el uso eficiente del buffer cache.
- **Importancia**: Clave para la gestión del rendimiento de la base de datos, asegurando que las escrituras de datos sean manejadas de manera eficiente para optimizar el acceso y el almacenamiento.

4. **LGWR (Log Writer)**:

- **Propósito**: Se encarga de escribir los registros de redo logs del buffer de redo log a los archivos de redo log en disco. Esto ocurre principalmente cuando una transacción se completa y se emite un COMMIT.
- **Importancia**: Crítico para la integridad de los datos, ya que asegura que todas las transacciones estén adecuadamente registradas y se puedan recuperar en caso de fallo.

5. **CKPT (Checkpoint)**:

- **Propósito**: El proceso de checkpoint señala el punto hasta el cual los datos han sido escritos en los archivos de datos desde los buffers, lo que ayuda en el proceso de recuperación para indicar desde dónde se debe empezar a aplicar los redo logs.
- **Importancia**: Reduce el tiempo de recuperación al limitar el número de operaciones de redo log que deben ser procesadas tras un reinicio.

6. **ARCH (Archiver)**:

- **Propósito**: Se encarga de copiar los redo log files que han sido llenados al almacenamiento de archivo o a una ubicación de backup. Estos archivos se conservan para la recuperación a largo plazo.
- **Importancia**: Vital para estrategias de recuperación de desastres y para mantener la continuidad del negocio, asegurando que los datos puedan ser restaurados a un estado anterior.

7. **RECO (Recoverer)**:

- **Propósito**: Automatiza la resolución de fallos en transacciones distribuidas. El proceso RECO intenta reconectar y completar transacciones pendientes entre nodos interconectados que podrían haberse interrumpido.
- **Importancia**: Clave para la gestión de transacciones en entornos distribuidos, asegurando que las transacciones pendientes se completen correctamente.

8. **LCK (Lock Manager)**:

- **Propósito**: Gestiona los bloqueos a nivel de base de datos que son necesarios para controlar el acceso concurrente a los datos y asegurar la integridad de los mismos.
- **Importancia**: Fundamental para el control de la concurrencia, permitiendo que múltiples usuarios trabajen en la base de datos simultáneamente sin interferir negativamente entre sí.

2.2 Estructura fisica de la base de datos.

La estructura física de una base de datos se refiere a cómo los datos se almacenan en el sistema de archivos del servidor de la base de datos. En el contexto de Oracle, la estructura física se compone de varios tipos de archivos que cumplen funciones específicas:

- **Datafiles (Archivos de datos)**: Son los archivos físicos donde se almacenan los objetos que forman parte de un tablespace. Un datafile pertenece solo a un tablespace y a una instancia de base de datos. Un tablespace puede estar formado por uno o varios datafiles. Al crear un datafile, se debe especificar su nombre, ubicación, tamaño y al tablespace al que pertenecerá. Estos archivos se llenan gradualmente a medida que se crean objetos en el tablespace, como tablas e índices, y se insertan registros en estas tablas 14.
- **Redo log files (Archivos de rehacer)**: Su propósito principal es la recuperación de datos a un momento en el tiempo o complementar una restauración de copia de respaldo completa. No contienen páginas, sino entradas con todos los cambios realizados en la base de datos, como modificaciones de datos, modificaciones de la base de datos y eventos de copia de seguridad y restauración. El acceso a datos es secuencial, ya que el registro de transacciones se actualiza en el mismo orden cronológico en el que se hacen las modificaciones 14.
- **Control files (Archivos de control)**: Son necesarios para mantener la integridad de la base de datos. Estos archivos contienen información sobre los datafiles y redo log files, permitiendo al sistema de base de datos saber cómo acceder a los datos almacenados 4.

Además de estos archivos principales, existen archivos externos o auxiliares como:

- **Archivos de parámetros**: Definen algunas características de una instancia de Oracle.
- **Archivos de contraseñas**: Sirven para autenticar a los usuarios.
- **Copias de archivos rehacer**: Utilizados para recuperar datos 4.

2.3 Requerimientos para la instalación.

2.4 Instalación del SGBD en modo transaccional

Las transacciones son un aspecto fundamental de los sistemas de gestión de bases de datos que aseguran la integridad de los datos y permiten el manejo seguro de errores y otros problemas que pueden surgir durante la ejecución de operaciones que modifican la base de datos. En MySQL, el manejo de transacciones permite a los desarrolladores agrupar varias tareas en una única unidad de trabajo que se ejecuta de manera integral, es decir, todas las tareas dentro de la transacción se completan con éxito, o si alguna falla, todas las operaciones previas se revierten a su estado original.

Propiedades de las Transacciones (ACID)
Las transacciones en bases de datos están diseñadas para ser:

- **Atómicas**: La transacción se realiza en su totalidad o no se realiza en absoluto.
- **Consistentes**: La transacción transforma la base de datos de un estado válido a otro estado válido.
- **Aisladas**: Los efectos de una transacción no son visibles para otras transacciones hasta que la transacción es confirmada.
- **Duraderas**: Una vez que la transacción ha sido confirmada, sus efectos son permanentes, incluso en caso de fallos del sistema.

Uso de Transacciones en MySQL
Para usar transacciones en MySQL, primero debes asegurarte de que estás usando un motor de almacenamiento que soporte transacciones, como InnoDB (el más común y recomendado para operaciones transaccionales).

Aquí te muestro cómo manejar transacciones en MySQL con algunos comandos básicos:

1. **BEGIN o START TRANSACTION**: Inicia una transacción.

```
1 START TRANSACTION;
```

2. **COMMIT**: Confirma los cambios realizados durante la transacción. Esto hará que los cambios sean permanentes.

```
1 COMMIT;
```

3. **ROLLBACK**: Revierte todos los cambios realizados en la base de datos desde el inicio de la transacción.

```
1 ROLLBACK;
```

4. **SAVEPOINT**: Establece un punto de salvado dentro de una transacción, al cual puedes revertir sin afectar el resto de la transacción.

```
1 SAVEPOINT savepoint_name;
```

5. **ROLLBACK TO SAVEPOINT**: Revierte la transacción al punto de salvado especificado.

```
1 ROLLBACK TO SAVEPOINT savepoint_name;
```

6. **RELEASE SAVEPOINT**: Elimina un punto de salvado.

```
1 RELEASE SAVEPOINT savepoint_name;
```

Ejemplo Práctico

Supongamos que quieres actualizar los balances de dos cuentas en un sistema bancario en respuesta a una transferencia de dinero:

```
START TRANSACTION;

-- Supone que retiras 100 de la cuenta 101
UPDATE cuentas SET balance = balance - 100 WHERE id = 101;

-- Y lo depositas en la cuenta 202
UPDATE cuentas SET balance = balance + 100 WHERE id = 202;

-- Verificar que no hay errores y confirmar la transacción
COMMIT;
```

En este ejemplo, si ocurre un error en alguna de las operaciones de actualización, podrías usar ROLLBACK para revertir todos los cambios y mantener la integridad de los datos:

```
ROLLBACK;
```

Consideraciones

- Siempre prueba tus transacciones en un entorno de desarrollo antes de implementarlas en producción para entender completamente su comportamiento.
- Monitorea el rendimiento, ya que las transacciones, especialmente las largas, pueden bloquear tablas y afectar el rendimiento general de la base de datos.

2.5 Variables de ambiente y archivos importantes para la instalación.

2.6 Procedimiento general para la instalación

2.7 Comandos Generales de alta y baja del sGBD